TIPURI DE DATE

Un **tip de dată** reprezintă o mulțime de obiecte împreună cu un set de operații care crează, distrug şi modifică aceste obiecte.

Sistemul de tipuri:

- **Tipuri predefinite**
- **Tipuri definite de programator** (scalare, structurate, pointer)

Sistemul de tipuri corespunde unui anumit limbaj.

TIPURI PREDEFINITE.

- **Tipuri numerice de bază:** întreg şi real
  Algol 68, Pascal : real
  Ada : float
- **Tipul logic:** un tip *enumerare* cu valori predefinite *true* şi *false*.
  Algol 68 : bool
  Ada, Pascal : boolean
- **Tipul caracter** – mulțimea caracterelor
  Algol 68, Pascal : char
  Ada : character

TIPURI DEFINITE DE PROGRAMATOR.

Definiția de tip constă în descrierea noului tip şi asocierea unui nume.

type *tab =* ***array*** *[1..10]* ***of*** *integer ;*
sau
var *t :* ***array*** *[1..10]* ***of*** *integer ;*

Tipuri scalare

Obiectele unui tip scalar sunt constante simple, care nu sunt compuse la rândul lor din alte valori (componente).

- Tipul enumerare
  type *zile = (luni,marti,miercuri,joi,vineri,sambata,duminica);*

- Tipuri numerice noi
  type *eps* ***is digits*** *10;*

- Subdomenii

PASCAL

type *zi_de_lucru = luni .. vineri ;*
litere_mici = 'a' .. 'z' ;
index = 0 .. 90 ;

ADA

type *eps_1 is* ***new*** *eps* ***range*** *–1.0 .. 1.0 ;*

Tipuri de date structurate

a) mecanismele de structurare
b) mecanismele de selecție

1. Produsul cartezian (articolul)

T = C1 x C2 x … x Cn

Fiecare element din T este de forma (a1,a2, … , an) cu ai de tip Ci

Ada, Pascal : ***record***
Algol 68, C : ***structure***

Exemplu Ada:

```
Type complex is
        record
                re, im : real
        end record ;
------------
var   c : complex ;
------------
c.re := 1 ;
c.im := 0 ;
-----------
```

sau *c := (1,0) ;*

2. Proiecția finită (tabloul)

Reprezintă o funcție definită pe o mulțime finită TI, cu valori într-o mulțime TE.

- Tablou:
  TI: tipul indicelui
  TE: tipul elementelor
  var *a :* ***array*** *[1 .. 100]* ***of*** *char ;*

  Elementele se selectează prin indexare: *a[k]*
  În Algol 68 şi Ada:
  a[11..20]:= ('0','1','2','3','4','5','6','7','8','9');

  Fixarea mulțimii indicilor (dimensiune tabloului):
  1. Fixarea dimensiunii indicilor la compilare (Fortran, C, Pascal).
  2. Fixarea mulțimii indicilor la execuție, în momentul creării variabilei (Algol 60, Ada).
  3. Tablouri cu dimensiune flexibilă pe parcursul executării programului (SNOBOL 4, Algol 68).

3. Secvența

Este o structură formată dintr-un număr oarecare de componente, toate de acelaşi tip.

- Fişierul secvențial (file)
- Sirul de caractere (string) (PL/I, Ada, Basic, Turbo Pascal)

4. Recursiunea (lista şi arborele)

Un tip T este recursiv dacă el are componente dintre care cel puțin una este de același tip T.

```
type arbore = record
                  informație : tip_info ;
                  subarb_st, subarb_dr : arbore;
              end;
```

- Pointeri (C, Pascal, Ada, Algol 68)
  Un arbore binar este reprezentat printr-o mulțime de noduri, fiecare de tipul:

```
type nod_arbore = record
                      informație : tip_info ;
                      subarb_st, subarb_dr : ↑nod_arbore
                  end ;
```

5. Reuniunile variabile

Permit specificarea unei structuri care, în mod opțional, poate lua una din mai multe forme alternative.

Forme diferite în diferite limbaje:

- C : **union**

```
union {
        float raza ;
       float l_dreptunghi [2];
       float l_triunghi [3];
} figura;
```

- Pascal, Ada : articol cu variante

```
type fig = (cerc, triunghi, dreptunghi);
     figura = record
          lungimea, aria : real;
          case forma : fig of
               cerc: (raza:real);
               dreptunghi: (l_drept:array[1..2]of real);
               triunghi: ( l_trg:array[1..3]of real;
                    dreptunghic:boolean)
          end;
```

6. Mulțimea

Variabilele de tipul **multime (T)** pot avea ca valoare orice submulțime a mulțimii corespunzătoare tipului T.

- În Pascal: set
- În alte limbaje poate fi implementată de programatori folosind tablouri, liste, arbori.

Tipul pointer

Un pointer reprezintă o referință către un anumit obiect. Este folosit pentru:

- implementarea structurilor de date recursive;
- transmitere de parametri (în C).

Probleme şi pericole potențiale:

- **Violarea compatibilităților de tip.**
- **Pseudonimele.**

```
var a,b : ↑t ;
 ---------
     a := new ( t );
     b := a ;
```

Obs: a şi b sunt pseudonime.

- **Referințe fictive.**

```
var a,b : ↑t ;
 ---------
     a := new ( t );
     b := a ;
 ---------
    dispose ( a ) ;
```

b rămâne o referință fictivă

```
---------------------------------
     int *p ;
     --------
     void f ( ) {
           int x ;
     ------------
     p := &x;
     ---------- }
 -----------
```

f ( ); după apelul funcției, *p* rămâne o referință fictivă

COMPATIBILITATEA TIPURILOR

Două tipuri T1şi T2 sunt compatibile, dacă :

- o valoare de tipul T1 poate fi atribuită unei variabile de tipul T2 (şi invers);
- un parametru actual de tipul T1 poate corespunde unuia formal de tipul T2 (şi invers).

```
type t = array [ 1..100 ] of integer;
     t1 = array [ 1..100 ] of integer;
    t2 = t1;
var a,b : array [ 1..100 ] of integer;
    c : t;
   d : t;
   e,f : t1;
  g : t2;
```

Modalități de definire a compatibilităților tipurilor:

1. Echivalența de nume (Ada)
   - Variabile de tip compatibil : *a* şi *b*, *c* şi *d*, *e* şi *f*;
   - Avantaj: simplitatea implementării.

2. Echivalența structurală (Algol 68, C)
   - Var. compatibile: *a,b,c,d,e,f,g*

```
type pret = integer;
          nr_studenti = integer;
    var cost : pret;
        efectiv : nr_studenti;
```

- *cost* şi *efectiv* sunt variabile compatibile conform echivalenței structurale şi sunt incompatibile în cazul echivalenței de nume.